Supervised Machine Learning

Badi Ahmed

November 22, 2025

Abstract

This paper offers a simple and accessible introduction to Supervised Learning, one of the most important and widely used approaches in Machine Learning. Following the basic ideas presented in our previous article, we explain how machines learn from labeled examples, similar to how a student learns with the help of a teacher. We walk through the main concepts, the most common algorithms, how models are evaluated, and where supervised learning is used in real life. The goal of this article is to give beginners a clear and practical understanding of how supervised learning works, and how it can be applied to solve everyday problems.

Keywords: Supervised Learning, Classification, Regression, Machine Learning Algorithms, Model Training, Model Evaluation, Predictive Modeling, Linear Regression, Logistic Regression, Decision Trees, Neural Networks

1 Introduction

In our previous article, we introduced the broader field of Machine Learning and saw that it is divided into several learning approaches. Among them, Supervised Learning is the most intuitive because it resembles how humans naturally learn: by looking at examples and receiving feedback. Just as a student practices exercises after seeing solutions from a teacher, a supervised model learns by observing many input–output pairs and gradually discovering the patterns that connect them. This simple idea makes supervised learning one of the most practical and widely used approaches in modern AI.

Today, supervised learning powers many technologies we use every day—from spam detection and voice recognition to medical diagnosis and recommendation systems. In this article, we explore how supervised learning works, the two main tasks it handles (classification and regression), the key algorithms behind it, and how to evaluate a model's performance. Our goal is to give readers a clear and practical understanding of supervised learning so they can recognize when and how to apply it to real-world problems.

1.1 What Makes Learning "Supervised"?

Learning is called *supervised* when the model is trained using labeled examples, meaning that both the input and the correct output are already known. During training, the model compares its predictions with the true answers and adjusts itself to reduce errors. This guidance from labeled data is what makes the learning process "supervised," as the model is shown exactly what it should learn.

1.2 The Teacher-Student Analogy

Supervised learning can be understood through a simple teacher-student analogy. A teacher provides example problems along with their correct solutions, and the student practices until they improve. In the same way, a supervised model receives many input–output pairs, makes

predictions, evaluates its mistakes, and gradually learns from them. This analogy mirrors the familiar learning process we experience in school.

2 How Supervised Learning Works

2.1 The Basic Process

At its core, supervised learning follows a remarkably straightforward process. Imagine you want to teach a child to recognize animals. You would show them pictures of different animals and tell them what each one is: "this is a dog," "this is a cat," "this is a bird." After seeing many examples, the child learns to recognize animals they have never seen before.

Supervised learning works exactly the same way, but with data and algorithms instead of pictures and children. The process can be broken down into four simple steps:

1. **Gather data:** Collect a dataset with inputs and correct outputs
2. **Choose algorithm:** Select the type of model to train
3. **Train model:** Feed labeled examples to learn patterns
4. **Test model:** Evaluate performance on new, unseen data

What makes this process "learning" rather than simple memorization is that the model discovers patterns and relationships in the data [1]. It does not just remember the training examples—it generalizes from them, building an internal representation that allows it to handle new situations.

The key insight is this: we do not program the rules explicitly. Instead, we let the model discover the rules by showing it many examples. This shift from explicit programming to learning from data is what makes machine learning fundamentally different from traditional software development [2].

2.2 Key Components

Every supervised learning system relies on four essential building blocks. Understanding these components is crucial because they form the foundation of everything else in supervised learning.

2.2.1 Features (Input Data X)

Features are the inputs to our model—the information we use to make predictions. Think of features as the questions we ask before making a decision.

If you are trying to predict whether it will rain tomorrow, your features might include today's temperature, humidity, wind speed, and cloud cover. If you are predicting house prices, your features might include the number of bedrooms, square footage, location, and age of the house.

Mathematically, we represent an input example as a vector:

$$\mathbf{x} = [x_1, x_2, x_3, \ldots, x_n] \tag{1}$$

where n is the number of features.

For a dataset with m examples, we organize all inputs into a matrix:

$$\mathbf{X} = \begin{bmatrix} x_1^{(1)} & x_2^{(1)} & \cdots & x_n^{(1)} \\ x_1^{(2)} & x_2^{(2)} & \cdots & x_n^{(2)} \\ \vdots & \vdots & \ddots & \vdots \\ x_1^{(m)} & x_2^{(m)} & \cdots & x_n^{(m)} \end{bmatrix} \tag{2}$$

Choosing the right features is often more important than choosing the right algorithm [3]. Good features capture the information that truly matters for your prediction task.

Features can be numerical (like age or salary), categorical (like color or city), or even more complex types like text or images. Part of the art of machine learning is transforming raw data into meaningful features that your model can learn from.

2.2.2 Labels (Target Output Y)

Labels are the correct answers—the outputs we want our model to predict. They represent what we are trying to learn.

In the animal recognition example, the labels would be "dog," "cat," "bird," and so on. In a house price prediction task, the labels would be the actual selling prices. In a medical diagnosis system, the labels might be "healthy" or "diseased."

For a single example, we denote the label as y. For a dataset of m examples, we have:

$$\mathbf{Y} = [y^{(1)}, y^{(2)}, \ldots, y^{(m)}]^T \tag{3}$$

Each example in our training dataset has both features and a corresponding label: $(\mathbf{x}^{(i)}, y^{(i)})$. This pairing of inputs with their correct outputs is what makes learning "supervised."

The type of label determines what kind of problem we have:

- **Classification:** Labels are discrete categories, $y \in \{1, 2, \ldots, K\}$
- **Regression:** Labels are continuous values, $y \in \mathbb{R}$

In the real world, obtaining labels can be expensive and time-consuming [4]. Someone needs to manually label thousands or millions of examples, which often requires expert knowledge. This labeling cost is one of the main limitations of supervised learning.

2.2.3 The Model (Function f)

The model is the heart of supervised learning—it is the mathematical function that transforms inputs into predictions. Mathematically, we write this as:

$$\hat{y} = f_\theta(\mathbf{x}) \tag{4}$$

where $\hat{y}$ represents our prediction and θ represents the model's parameters.

You can think of the model as a black box that takes features as input and produces a prediction as output. But what exactly is this function? It depends on the algorithm we choose.

For example, a simple linear regression model has the form:

$$\hat{y} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n = \theta_0 + \sum_{j=1}^{n} \theta_j x_j \tag{5}$$

In vector notation, this becomes:

$$\hat{y} = \boldsymbol{\theta}^T \mathbf{x} \tag{6}$$

The model contains parameters—internal values that determine exactly how it transforms inputs into outputs [5]. These parameters start with random values, and the entire point of training is to adjust them until the model makes good predictions.

What makes a model powerful is its ability to capture complex patterns. Simple models like linear regression can only learn simple relationships. More complex models like neural networks can learn intricate, non-linear patterns.

2.2.4 Loss Function

The loss function measures how wrong the model is. It compares the model's predictions to the true labels, producing a single number that quantifies the error.

For a single example, the loss is denoted as $L(\hat{y}, y)$. For the entire dataset, we calculate the average loss:

$$J(\boldsymbol{\theta}) = \frac{1}{m} \sum_{i=1}^{m} L(\hat{y}^{(i)}, y^{(i)}) \tag{7}$$

Different problems require different loss functions. For regression, we commonly use Mean Squared Error (MSE):

$$J(\boldsymbol{\theta}) = \frac{1}{m} \sum_{i=1}^{m} (\hat{y}^{(i)} - y^{(i)})^2 \tag{8}$$

For binary classification, we use Binary Cross-Entropy:

$$J(\boldsymbol{\theta}) = -\frac{1}{m} \sum_{i=1}^{m} \left[y^{(i)} \log(\hat{y}^{(i)}) + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)}) \right] \tag{9}$$

The loss function is crucial because it guides the learning process [6]. During training, the algorithm's entire goal is to minimize the loss. It does this by adjusting the model's parameters to make that error as small as possible.

Think of the loss as a report card that grades the model's performance. A high loss means many or large mistakes. A low loss means the model is doing well.

2.3 The Learning Loop

Now that we understand the key components, let us see how they all work together. Training a supervised learning model is an iterative process—a loop that repeats thousands or millions of times until the model learns to make good predictions.

The training loop follows these steps:

1. **Forward Pass:** Feed training examples through the model to get predictions
2. **Compute Loss:** Calculate how wrong the predictions are
3. **Backward Pass:** Compute gradients of the loss with respect to parameters
4. **Update Parameters:** Adjust parameters to reduce the loss
5. **Repeat:** Continue until convergence

The key question is: how should we adjust the parameters to reduce the loss? This is where gradient descent comes in [7]. We calculate the gradient (derivative) of the loss with respect to each parameter:

$$\nabla_\theta J(\boldsymbol{\theta}) = \left[\frac{\partial J}{\partial \theta_1}, \frac{\partial J}{\partial \theta_2}, \ldots, \frac{\partial J}{\partial \theta_n} \right] \tag{10}$$

Then we update each parameter by taking a small step in the opposite direction of the gradient:

$$\theta_j := \theta_j - \alpha \frac{\partial J}{\partial \theta_j} \tag{11}$$

where α is the learning rate—a small positive number that controls the step size.

Think of it like being lost in a foggy mountain and trying to reach the valley below. You feel the slope beneath your feet and take a small step downhill. That is what gradient descent

does: it computes the slope of the loss function and takes a small step in the direction that leads downward.

In practice, we often use mini-batch gradient descent, where we update parameters using a subset of the data:

$$\boldsymbol{\theta} := \boldsymbol{\theta} - \alpha \cdot \frac{1}{b} \sum_{i=1}^{b} \nabla_\theta L(\hat{y}^{(i)}, y^{(i)}) \tag{12}$$

where b is the batch size.

With each iteration, the model gets slightly better. The loss gradually decreases, and the predictions become more accurate. We typically stop when the loss stops decreasing significantly, which indicates convergence.

The beauty of this learning loop is its generality. Whether we are training a simple linear regression or a massive neural network with billions of parameters, the fundamental process remains the same: predict, measure error, adjust parameters, repeat [8].

One final note: the learning loop requires careful tuning. We need to decide how big each step should be (learning rate α), how many examples to process at once (batch size b), and when to stop training. These choices, called hyperparameters, can significantly affect whether the model learns successfully.

3 The Two Main Tasks in Supervised Learning

Supervised learning solves two fundamental types of problems: classification and regression. The distinction between them is simple but important. Classification predicts categories or classes—discrete labels like "yes" or "no," "cat" or "dog." Regression predicts continuous numerical values like prices, temperatures, or distances.

Understanding which type of problem you have is crucial because it determines everything else: which algorithms to use, which loss functions to optimize, and which metrics to evaluate. Let us explore each type in detail.

3.1 Classification: Predicting Categories

3.1.1 What is Classification?

Classification is the task of assigning inputs to predefined categories or classes. Think of it as sorting items into labeled boxes. Given an input example, the model must decide which category it belongs to.

Mathematically, in classification, our target variable y takes on discrete values from a finite set of classes:

$$y \in \{C_1, C_2, \ldots, C_K\} \tag{13}$$

where K is the number of classes.

For example, in email spam detection, we have two classes:

$$y \in \{\text{spam}, \text{not spam}\} \tag{14}$$

The model learns a decision boundary that separates different classes in the feature space [5]. For a binary classification problem, this boundary divides the space into two regions. For multi-class problems, we have multiple boundaries separating different classes.

The output of a classification model is typically either:

- A class label: $\hat{y} = C_k$
- Class probabilities: $P(y = C_k|\mathbf{x})$ for each class k

The probabilistic interpretation is often more useful because it tells us not just what the model predicts, but how confident it is in that prediction [9].

3.1.2 Types of Classification

Classification problems come in different varieties based on the number of classes involved. Let us examine the main types.

Binary Classification

Binary classification involves exactly two classes. We typically encode these as $y \in \{0, 1\}$ or $y \in \{-1, +1\}$. This is the simplest and most common type of classification.

The model learns a function that maps inputs to one of two outcomes:

$$f(\mathbf{x}) = \begin{cases} 1 & \text{if } P(y = 1|\mathbf{x}) \geq 0.5 \\ 0 & \text{otherwise} \end{cases} \tag{15}$$

Common examples include:

- Spam detection (spam vs. not spam)
- Disease diagnosis (diseased vs. healthy)
- Loan approval (approve vs. reject)
- Churn prediction (will leave vs. will stay)

For binary classification, we often use logistic regression, which models the probability as:

$$P(y = 1|\mathbf{x}) = \sigma(\boldsymbol{\theta}^T \mathbf{x}) = \frac{1}{1 + e^{-\boldsymbol{\theta}^T \mathbf{x}}} \tag{16}$$

where σ is the sigmoid function [10].

Multi-class Classification

Multi-class classification involves three or more classes. Each example belongs to exactly one class out of K possible classes.

We represent the target as $y \in \{1, 2, \ldots, K\}$. The model must decide which single class the input belongs to.

Examples include:

- Digit recognition: classifying handwritten digits 0-9 ($K = 10$)
- Animal classification: cat, dog, bird, fish, etc.
- Language identification: English, French, Spanish, etc.
- Product categorization: electronics, clothing, books, etc.

For multi-class problems, we often use the softmax function to compute class probabilities:

$$P(y = k|\mathbf{x}) = \frac{e^{\boldsymbol{\theta}_k^T \mathbf{x}}}{\sum_{j=1}^{K} e^{\boldsymbol{\theta}_j^T \mathbf{x}}} \tag{17}$$

The predicted class is then:

$$\hat{y} = \arg\max_k P(y = k|\mathbf{x}) \tag{18}$$

Multi-label Classification

Multi-label classification is a special case where each example can belong to multiple classes simultaneously. This is different from multi-class classification where each example belongs to exactly one class.

Formally, for K possible labels, the output is a binary vector:

$$\mathbf{y} = [y_1, y_2, \ldots, y_K] \text{ where } y_k \in \{0, 1\} \tag{19}$$

Examples include:

- Image tagging: a photo might contain both "beach" and "sunset" and "people"
- Movie genres: a film can be both "action" and "comedy"
- News article categorization: an article might be tagged "politics," "economy," and "international"

Multi-label problems are typically solved by training K independent binary classifiers, one for each label [11].

3.2 Regression: Predicting Numbers

3.2.1 What is Regression?

Regression is the task of predicting continuous numerical values. Unlike classification, where we choose from discrete categories, regression produces numbers that can take any value within a range.

Mathematically, in regression, our target variable is continuous:

$$y \in \mathbb{R} \tag{20}$$

The model learns a function that maps input features to a real-valued output:

$$\hat{y} = f_\theta(\mathbf{x}) \tag{21}$$

The simplest form is linear regression, where we assume a linear relationship:

$$\hat{y} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n \tag{22}$$

In matrix form:

$$\hat{\mathbf{y}} = \mathbf{X}\boldsymbol{\theta} \tag{23}$$

The goal is to find parameters $\boldsymbol{\theta}$ that minimize the prediction error. We typically use Mean Squared Error as the loss function:

$$J(\boldsymbol{\theta}) = \frac{1}{m}\sum_{i=1}^{m}(\hat{y}^{(i)} - y^{(i)})^2 \tag{24}$$

For linear regression, we can solve this analytically using the normal equation [12]:

$$\boldsymbol{\theta} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y} \tag{25}$$

Regression assumes that the relationship between features and target can be approximated by a smooth function. The model interpolates between training examples to make predictions for new inputs.

3.3 Classification vs Regression: Key Differences

Understanding the distinction between classification and regression is essential for choosing the right approach. Let us summarize the key differences:

Aspect	Classification	Regression
Output type	Discrete categories	Continuous numbers
Examples	Spam/not spam, cat/dog/bird	Price, temperature, distance
Common loss	Cross-entropy	Mean Squared Error (MSE)
Evaluation metrics	Accuracy, precision, recall, F1	MSE, MAE, R^2
Typical algorithms	Logistic regression, decision trees, SVM	Linear regression, polynomial regression
Output interpretation	Class label or probability	Numerical value

Table 1: Comparison between Classification and Regression tasks

Sometimes the line between classification and regression can blur. For example, predicting house prices is clearly regression. But if we instead predict whether a house is "expensive" or "affordable," that becomes classification. The underlying data is the same—only our formulation of the problem changes.

Some problems can be approached either way:

- Age prediction: regression (predict exact age) or classification (predict age group)
- Customer satisfaction: regression (predict rating 1-5) or classification (satisfied/unsatisfied)
- Risk assessment: regression (predict risk score) or classification (high/medium/low risk)

Choosing between classification and regression depends on your specific needs. Classification is better when you need clear categories for decision-making. Regression is better when you need precise numerical estimates and the exact value matters [13].

4 Common Supervised Learning Algorithms

Supervised learning offers a rich toolkit of algorithms, each with its own strengths and ideal use cases. In this section, we provide a brief overview of the most popular algorithms. Detailed explanations, mathematical derivations, and implementation examples will be covered in separate dedicated articles.

The choice of algorithm depends on several factors: the nature of your data, the size of your dataset, the complexity of the problem, interpretability requirements, and computational resources available [13].

4.1 Linear Regression

Linear regression is the simplest and most widely used regression algorithm. It models the relationship between input features and a continuous target variable using a linear equation.

Key Idea: Find the best straight line (or hyperplane) that fits the data:

$$\hat{y} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n \tag{26}$$

Best for: Predicting continuous values when the relationship between features and target is approximately linear (house prices, temperatures, sales).

Advantages: Fast, interpretable, requires little training data, mathematically well-understood.

Limitations: Only captures linear relationships, sensitive to outliers, assumes features are independent.

4.2 Logistic Regression

Despite its name, logistic regression is a classification algorithm. It predicts the probability that an input belongs to a particular class using the sigmoid function.

Key Idea: Model probability using:

$$P(y = 1|x) = \frac{1}{1 + e^{-(\theta_0 + \theta_1 x_1 + \cdots + \theta_n x_n)}} \tag{27}$$

Best for: Binary classification problems (spam detection, disease diagnosis, customer churn).

Advantages: Outputs probabilities, fast training, works well with linearly separable data, interpretable coefficients [10].

Limitations: Assumes linear decision boundary, struggles with complex non-linear patterns, requires feature engineering for interactions.

4.3 Decision Trees

Decision trees create a tree-like model of decisions based on feature values. Each internal node represents a test on a feature, each branch represents an outcome, and each leaf represents a class label or value.

Key Idea: Recursively split data based on features that best separate classes or reduce variance:

$$\text{Split on feature } x_j \text{ that maximizes information gain or reduces MSE} \tag{28}$$

Best for: Both classification and regression, especially when relationships are non-linear and interactions between features matter.

Advantages: Highly interpretable (can visualize the tree), handles non-linear relationships, requires minimal data preprocessing, captures feature interactions naturally [14].

Limitations: Prone to overfitting, unstable (small data changes cause large tree changes), biased toward features with many levels.

4.4 Random Forests

Random forests are ensemble models that combine multiple decision trees. Each tree is trained on a random subset of data and features. The final prediction aggregates all individual tree predictions.

Key Idea: "Wisdom of the crowd" - average many trees:

$$\hat{y} = \frac{1}{B} \sum_{b=1}^{B} \hat{y}_b \tag{29}$$

where B is the number of trees.

Best for: Complex problems requiring high accuracy with less risk of overfitting than single trees.

Advantages: Highly accurate, reduces overfitting through averaging, handles missing values, provides feature importance, works well out-of-the-box [15].

Limitations: Less interpretable than single trees, slower training and prediction, requires more memory, can be overkill for simple problems.

4.5 Support Vector Machines (SVM)

SVM finds the optimal hyperplane that maximally separates classes in feature space. It focuses on the most difficult examples (support vectors) near the decision boundary.

Key Idea: Maximize the margin between classes:

$$\max_{\theta,\theta_0} \frac{2}{\|\theta\|} \text{ subject to } y^{(i)}(\theta^T x^{(i)} + \theta_0) \geq 1 \tag{30}$$

SVM can handle non-linear boundaries using the kernel trick:

$$K(x^{(i)}, x^{(j)}) = \phi(x^{(i)})^T \phi(x^{(j)}) \tag{31}$$

Best for: Binary classification, especially with clear margins between classes, high-dimensional data (text classification, image recognition).

Advantages: Effective in high dimensions, memory efficient (uses only support vectors), versatile through different kernel functions [16].

Limitations: Slow for large datasets, requires careful parameter tuning, difficult to interpret, struggles with noisy data and overlapping classes.

4.6 K-Nearest Neighbors (KNN)

KNN is a simple, instance-based algorithm. It classifies new examples based on the majority class of their k nearest neighbors in the feature space.

Key Idea: "You are the average of your neighbors":

$$\hat{y} = \text{mode}\{y^{(i)} : x^{(i)} \in \text{k-nearest neighbors of } x\} \tag{32}$$

Distance is typically Euclidean:

$$d(x^{(i)}, x^{(j)}) = \sqrt{\sum_{l=1}^{n} (x_l^{(i)} - x_l^{(j)})^2} \tag{33}$$

Best for: Small to medium datasets, problems where similar examples should have similar outputs, recommendation systems.

Advantages: Simple to understand and implement, no training phase (lazy learning), naturally handles multi-class problems, non-parametric (makes no assumptions about data distribution) [17].

Limitations: Slow prediction for large datasets, sensitive to irrelevant features and feature scaling, requires choosing appropriate k value, struggles in high dimensions (curse of dimensionality).

4.7 Gradient Boosting

Gradient boosting builds an ensemble by sequentially adding weak learners (usually shallow decision trees) that correct the errors of previous learners. Each new tree focuses on the mistakes of the ensemble so far.

Key Idea: Iteratively improve by fitting to residual errors:

$$F_m(x) = F_{m-1}(x) + \alpha \cdot h_m(x) \tag{34}$$

where h_m is trained on the residuals of F_{m-1}.

Best for: Structured/tabular data competitions, problems requiring highest possible accuracy, Kaggle competitions.

Advantages: State-of-the-art accuracy on many problems, handles mixed data types, automatic feature interaction detection, provides feature importance [18].

Popular implementations: XGBoost, LightGBM, CatBoost.

Limitations: Requires careful hyperparameter tuning, prone to overfitting if not regularized, slower training than random forests, less interpretable, computationally intensive.

4.8 Naive Bayes

Naive Bayes applies Bayes' theorem with the "naive" assumption that features are conditionally independent given the class label. Despite this strong assumption, it often works surprisingly well.

Key Idea: Use Bayes' theorem:

$$P(y|x_1, \ldots, x_n) = \frac{P(y) \prod_{i=1}^{n} P(x_i|y)}{P(x_1, \ldots, x_n)} \tag{35}$$

Predict the class with highest posterior probability:

$$\hat{y} = \arg\max_y P(y) \prod_{i=1}^{n} P(x_i|y) \tag{36}$$

Best for: Text classification (spam filtering, sentiment analysis), problems with categorical features, when training data is limited.

Advantages: Very fast training and prediction, works well with small datasets, handles high-dimensional data, simple and interpretable, performs well even when independence assumption is violated [19].

Limitations: Independence assumption rarely holds in practice, cannot learn feature interactions, sensitive to how features are encoded, probability estimates can be unreliable.

4.9 Neural Networks

Neural networks are composed of layers of interconnected nodes (neurons) that transform inputs through non-linear activations. Deep networks with many layers can learn complex hierarchical representations.

Key Idea: Stack multiple layers of transformations:

$$h^{(l)} = \sigma(W^{(l)} h^{(l-1)} + b^{(l)}) \tag{37}$$

where σ is a non-linear activation function (ReLU, sigmoid, tanh).

Best for: Complex pattern recognition (images, speech, text), problems with large amounts of data, feature learning from raw inputs.

Advantages: Can approximate any function, automatically learns features, state-of-the-art on many tasks (computer vision, NLP), scales well with data and compute [6].

Limitations: Requires large amounts of training data, computationally expensive, many hyperparameters to tune, black-box (hard to interpret), prone to overfitting without regularization.

5 Conclusion

Supervised learning is like learning with a teacher: the machine receives examples with the correct answers and practices predicting new situations. This kind of learning is everywhere in our lives (spam filtering, voice recognition, price prediction, etc.). The main idea: we don't program all the rules by hand; we let the machine discover them from data. There are several methods depending on what we want to predict, but they all work by comparing predictions with reality and improving step by step. In summary, supervised learning lets machines learn from examples, and it's the reason behind many of today's advances in artificial intelligence.

References

[1] T. M. Mitchell, *Machine Learning*. McGraw-Hill, 1997.

[2] A. L. Samuel, "Some studies in machine learning using the game of checkers," *IBM Journal of Research and Development*, vol. 3, no. 3, pp. 210–229, 1959.

[3] P. Domingos, "A few useful things to know about machine learning," *Communications of the ACM*, vol. 55, no. 10, pp. 78–87, 2012.

[4] B. Settles, "Active learning literature survey," University of Wisconsin-Madison, Tech. Rep. 1648, 2009.

[5] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning*, 2nd. Springer, 2009.

[6] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.

[7] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, pp. 533–536, 1986.

[8] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, pp. 436–444, 2015.

[9] K. P. Murphy, "Machine learning: A probabilistic perspective," 2012.

[10] D. W. Hosmer, S. Lemeshow, and R. X. Sturdivant, *Applied Logistic Regression*, 3rd. Wiley, 2013.

[11] G. Tsoumakas and I. Katakis, "Multi-label classification: An overview," *International Journal of Data Warehousing and Mining*, vol. 3, no. 3, pp. 1–13, 2007.

[12] J. Friedman, T. Hastie, and R. Tibshirani, *The Elements of Statistical Learning*. Springer, 2001.

[13] S. B. Kotsiantis, I. Zaharakis, and P. Pintelas, "Supervised machine learning: A review of classification techniques," *Emerging Artificial Intelligence Applications in Computer Engineering*, vol. 160, pp. 3–24, 2007.

[14] L. Breiman, J. Friedman, C. J. Stone, and R. A. Olshen, *Classification and Regression Trees*. CRC Press, 1984.

[15] L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.

[16] C. Cortes and V. Vapnik, "Support-vector networks," *Machine Learning*, vol. 20, no. 3, pp. 273–297, 1995.

[17] T. Cover and P. Hart, "Nearest neighbor pattern classification," *IEEE Transactions on Information Theory*, vol. 13, no. 1, pp. 21–27, 1967.

[18] J. H. Friedman, "Greedy function approximation: A gradient boosting machine," *Annals of Statistics*, vol. 29, no. 5, pp. 1189–1232, 2001.

[19] I. Rish, "An empirical study of the naive bayes classifier," in *IJCAI Workshop on Empirical Methods in Artificial Intelligence*, vol. 3, 2001, pp. 41–46.